Showing that Android’s, Java’s and Python’s sorting algorithm is broken and proving a fixed version

Stijn de Gouw
Jurriaan Rot, Frank de Boer, Richard Bubel, Reiner Hähnle

Open University

Studiedag Informatica
Eindhoven, November 24, 2018
### Library

Collection of algorithms with a well-defined interface that perform a commonly used task
Library

Collection of algorithms with a well-defined interface that perform a commonly used task

Example: Java standard library functions

Programming to interfaces:

- Sorting a given array `a`
  ```java
  static void sort(Object[] a)
  ```

- Searching a value `key` in the array `a`
  ```java
  static int binarySearch(Object[] a, Object key)
  ```

Usability of programming language partially depends on good libraries
Library

Collection of algorithms with a well-defined interface that perform a commonly used task

Example: Java standard library functions

Programming to interfaces:

- Sorting a given array `a`
  ```java
  static void sort(Object[] a)
  ```

- Searching a value `key` in the array `a`
  ```java
  static int binarySearch(Object[] a, Object key)
  ```

Usability of programming language partially depends on good libraries

Correctness of library functions is crucial:
used as building blocks in millions of programs
Timsort: Bird’s view

Tim Peters — “hey, I earned it <wink>”

TimSort description
A hybrid sorting algorithm (insertion sort + merge sort) optimized for partially sorted arrays (often encountered in real-world data).
TimSort: Bird’s view

Tim Peters — “hey, I earned it <wink>”

TimSort description

A hybrid sorting algorithm (insertion sort + merge sort) optimized for partially sorted arrays (often encountered in real-world data).

Why analyze Timsort?

▶ Complex algorithm, widely used
▶ Extensively tested + manual code reviews: bugs unlikely!?
Main loop

- Find next run (extending to length $\geq 16$ with insertion sort), add length to runLen array

- Merge until last run lengths in runLen satisfy:
  1. $X > Y + Z$
  2. $Y > Z$

Merge if (1) or (2) is false:
- if (1) is false and $X < Z$: merge $X$ and $Y$
- otherwise: merge $Y$ and $Z$

At the end: merge all runs, resulting in a sorted array
Main loop

- Find next run (extending to length $\geq 16$ with insertion sort), add length to runLen array
- Merge until last run lengths in runLen satisfy:
  1. $X > Y + Z$
  2. $Y > Z$

Merge if (1) or (2) is false:
- if (1) is false and $X < Z$: merge $X$ and $Y$
- otherwise: merge $Y$ and $Z$

At the end: merge all runs, resulting in a sorted array

Example (ignoring length $\geq 16$ requirement)

<table>
<thead>
<tr>
<th>Input</th>
<th>1 2 3 4 5 0 1 1 0 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>runLen</td>
<td></td>
</tr>
</tbody>
</table>
Timsort: the algorithm

Main loop

- **Find next run (extending to length ≥ 16 with insertion sort)**, add length to runLen array
- Merge until last run lengths in runLen \[ \ldots \ X \ Y \ Z \] satisfy:
  1. \( X > Y + Z \)
  2. \( Y > Z \)

Merge if (1) or (2) is false:
- if (1) is false and \( X < Z \): merge \( X \) and \( Y \)
- otherwise: merge \( Y \) and \( Z \)

At the end: merge all runs, resulting in a sorted array

Example (ignoring length ≥ 16 requirement)

<table>
<thead>
<tr>
<th>Input</th>
<th>1 2 3 4 5 0 1 1 0 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>runLen</td>
<td>5</td>
</tr>
</tbody>
</table>
Timsort: the algorithm

Main loop

▶ Find next run (extending to length $\geq 16$ with insertion sort), add length to runLen array

▶ Merge until last run lengths in runLen $\cdots X Y Z$ satisfy:

(1) $X > Y + Z$
(2) $Y > Z$

Merge if (1) or (2) is false:

• if (1) is false and $X < Z$: merge $X$ and $Y$
• otherwise: merge $Y$ and $Z$

At the end: merge all runs, resulting in a sorted array

Example (ignoring length $\geq 16$ requirement)

<table>
<thead>
<tr>
<th>Input</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>runLen</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(1) and (2) do not apply, not enough runs
Main loop

- **Find next run** (extending to length $\geq 16$ with insertion sort), add length to \texttt{runLen} array
- Merge until last run lengths in \texttt{runLen} satisfy:
  
  (1) $X > Y + Z$
  
  (2) $Y > Z$

  Merge if (1) or (2) is false:
  
  - if (1) is false and $X < Z$: merge $X$ and $Y$
  - otherwise: merge $Y$ and $Z$

At the end: merge all runs, resulting in a sorted array

**Example (ignoring length $\geq 16$ requirement)**

<table>
<thead>
<tr>
<th>Input</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{runLen}</td>
<td>5</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Main loop

- Find next run (extending to length $\geq 16$ with insertion sort), add length to `runLen` array

- Merge until last run lengths in `runLen` satisfy:
  1. $X > Y + Z$
  2. $Y > Z$

  Merge if (1) or (2) is false:
  - if (1) is false and $X < Z$: merge $X$ and $Y$
  - otherwise: merge $Y$ and $Z$

At the end: merge all runs, resulting in a sorted array

Example (ignoring length $\geq 16$ requirement)

<table>
<thead>
<tr>
<th>Input</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>runLen</code></td>
<td>5</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(1) does not apply, and (2) $5 > 3$
Main loop

- **Find next run (extending to length ≥ 16 with insertion sort)**, add length to runLen array
- Merge until last run lengths in runLen satisfy:
  1. \( X > Y + Z \)
  2. \( Y > Z \)

Merge if (1) or (2) is false:
- if (1) is false and \( X < Z \): merge X and Y
- otherwise: merge Y and Z

At the end: merge all runs, resulting in a sorted array

Example (ignoring length ≥ 16 requirement)

<table>
<thead>
<tr>
<th>Input</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>runLen</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Main loop

- Find next run (extending to length $\geq 16$ with insertion sort), add length to runLen array
- Merge until last run lengths in runLen satisfy:
  
  (1) $X > Y + Z$
  
  (2) $Y > Z$

Merge if (1) or (2) is false:
- if (1) is false and $X < Z$: merge $X$ and $Y$
- otherwise: merge $Y$ and $Z$

At the end: merge all runs, resulting in a sorted array

Example (ignoring length $\geq 16$ requirement)

<table>
<thead>
<tr>
<th>Input</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>runLen</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(1) $5 > 3 + 2$ does not hold!
Timsort: the algorithm

Main loop

- Find next run (extending to length $\geq 16$ with insertion sort), add length to runLen array

- **Merge until last run lengths in runLen** $\cdots X\ Y\ Z$ satisfy:
  1. $X > Y + Z$
  2. $Y > Z$

Merge if (1) or (2) is false:
- if (1) is false and $X < Z$: merge $X$ and $Y$
- otherwise: merge $Y$ and $Z$

At the end: merge all runs, resulting in a sorted array

Example (ignoring length $\geq 16$ requirement)

<table>
<thead>
<tr>
<th>Input</th>
<th>runLen</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 0 0 1 1 4</td>
<td>5 5</td>
</tr>
</tbody>
</table>
Main loop

- Find next run (extending to length $\geq 16$ with insertion sort), add length to runLen array

- **Merge until last run lengths in runLen** satisfy:
  1. $X > Y + Z$
  2. $Y > Z$

Merge if (1) or (2) is false:
- if (1) is false and $X < Z$: merge $X$ and $Y$
- otherwise: merge $Y$ and $Z$

At the end: merge all runs, resulting in a sorted array

Example (ignoring length $\geq 16$ requirement)

<table>
<thead>
<tr>
<th>Input</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>runLen</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(2) $5 > 5$ does not hold!
Main loop

- Find next run (extending to length ≥ 16 with insertion sort), add length to \( \text{runLen} \) array

- **Merge until last run lengths in \( \text{runLen} \) \( \cdot \cdot \cdot X Y Z \) satisfy:**
  
  1. \( X > Y + Z \)
  2. \( Y > Z \)

  Merge if (1) or (2) is false:
  
  - if (1) is false and \( X < Z \): merge \( X \) and \( Y \)
  - otherwise: merge \( Y \) and \( Z \)

At the end: merge all runs, resulting in a sorted array

Example (ignoring length ≥ 16 requirement)

<table>
<thead>
<tr>
<th>Input</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{runLen} )</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(1) and (2) do not apply, not enough runs
Main loop

- Find next run (extending to length $\geq 16$ with insertion sort), add length to runLen array

- Merge until last run lengths in runLen satisfy:
  1. $X > Y + Z$
  2. $Y > Z$

  Merge if (1) or (2) is false:
  - if (1) is false and $X < Z$: merge $X$ and $Y$
  - otherwise: merge $Y$ and $Z$

At the end: merge all runs, resulting in a sorted array

Example (ignoring length $\geq 16$ requirement)

<table>
<thead>
<tr>
<th>Input</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>runLen</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Breaking the invariant

Size of \( \text{runLen} \)

1. \( \text{runLen}[i-2] > \text{runLen}[i-1] + \text{runLen}[i] \)
2. \( \text{runLen}[i-1] > \text{runLen}[i] \)

If the above invariant is true for all \( i \) and \( \text{runLen}[i] \geq 16 \), then

- (reversed) runlengths grow exponentially fast (... 87 52 34 17 16)
- Runs do not overlap: few runs required to cover input array
Breaking the invariant

Size of \( \text{runLen} \)

1. \( \text{runLen}[i-2] > \text{runLen}[i-1] + \text{runLen}[i] \)
2. \( \text{runLen}[i-1] > \text{runLen}[i] \)

If the above invariant is true for all \( i \) and \( \text{runLen}[i] \geq 16 \), then

- (reversed) runlengths grow exponentially fast (... 87 \( 52+34+17+16 \))
- Runs do not overlap: few runs required to cover input array

```java
int len = a.length;
...
int stackLen = (len < 120 ? 4 :
                 len < 1542 ? 9 :
                 len < 119151 ? 18 : 39);
runBase = new int[stackLen];
runLen = new int[stackLen];
```
Breaking the invariant

Size of runLen

1. \( \text{runLen}[i-2] > \text{runLen}[i-1] + \text{runLen}[i] \)
2. \( \text{runLen}[i-1] > \text{runLen}[i] \)

If the above invariant is true for all \( i \) and \( \text{runLen}[i] \geq 16 \), then

- (reversed) runlengths grow exponentially fast (... 87 52 34 17 16)
- Runs do not overlap: few runs required to cover input array
Breaking the invariant

Size of runLen

1. \( \text{runLen}[i-2] > \text{runLen}[i-1] + \text{runLen}[i] \)
2. \( \text{runLen}[i-1] > \text{runLen}[i] \)

If the above invariant is true for all \( i \) and \( \text{runLen}[i] \geq 16 \), then

\( (\text{reversed}) \) runlengths grow exponentially fast (... 87 52 34 17 16)

Runs do not overlap: few runs required to cover input array

Breaking the invariant - checking last 3 runs is insufficient

Label last 3 runs as \( X, Y, Z \).
If \( X > Y + Z \) is false and \( X < Z \): merge \( X \) and \( Y \), otherwise \( Y \) and \( Z \)

| runLen | 120 | 80  | 25  | 20  |
Breaking the invariant

Size of runLen

1. \( \text{runLen}[i-2] > \text{runLen}[i-1] + \text{runLen}[i] \)
2. \( \text{runLen}[i-1] > \text{runLen}[i] \)

If the above invariant is true for all \( i \) and \( \text{runLen}[i] \geq 16 \), then

- (reversed) runlengths grow exponentially fast (\ldots 87 52 34 17 16)
- Runs do not overlap: few runs required to cover input array

Breaking the invariant - checking last 3 runs is insufficient

Label last 3 runs as \( X, Y, Z \).

If \( X > Y + Z \) is false and \( X < Z \): merge \( X \) and \( Y \), otherwise \( Y \) and \( Z \)

<table>
<thead>
<tr>
<th>runLen</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
</tr>
<tr>
<td>80</td>
</tr>
<tr>
<td>25</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>30</td>
</tr>
</tbody>
</table>
Breaking the invariant

Size of runLen

1. \( \text{runLen}[i-2] > \text{runLen}[i-1] + \text{runLen}[i] \)
2. \( \text{runLen}[i-1] > \text{runLen}[i] \)

If the above invariant is true for all \( i \) and \( \text{runLen}[i] \geq 16 \), then

- (reversed) runlengths grow exponentially fast (... 87 52 34 17 16)
- Runs do not overlap: few runs required to cover input array

Breaking the invariant - checking last 3 runs is insufficient

Label last 3 runs as \( X, Y, Z \).

**If** \( X > Y + Z \) **is false and** \( X < Z \): **merge** \( X \) **and** \( Y \), otherwise \( Y \) and \( Z \)

| runLen | 120 | 80  | 25  | 20  | 30  |
Breaking the invariant

Size of $\text{runLen}$

1. $\text{runLen}[i-2] > \text{runLen}[i-1] + \text{runLen}[i]$
2. $\text{runLen}[i-1] > \text{runLen}[i]$

If the above invariant is true for all $i$ and $\text{runLen}[i] \geq 16$, then

- (reversed) runlengths grow exponentially fast (... 87 52 34 17 16)
- Runs do not overlap: few runs required to cover input array

Breaking the invariant - checking last 3 runs is insufficient

Label last 3 runs as $X, Y, Z$.
If $X > Y + Z$ is false and $X < Z$: merge $X$ and $Y$, otherwise $Y$ and $Z$

<table>
<thead>
<tr>
<th>runLen</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
</tr>
<tr>
<td>80</td>
</tr>
<tr>
<td>45</td>
</tr>
<tr>
<td>30</td>
</tr>
</tbody>
</table>
Breaking the invariant

Size of \texttt{runLen}

1. \texttt{runLen[i-2] > runLen[i-1] + runLen[i]}
2. \texttt{runLen[i-1] > runLen[i]}

If the above invariant is true for all \(i\) and \(\text{runLen}[i] \geq 16\), then

- (reversed) runlengths grow exponentially fast (... 87 52 34 17 16)
- Runs do not overlap: few runs required to cover input array

Breaking the invariant - checking last 3 runs is insufficient

Label last 3 runs as \(X, Y, Z\).
If \(X > Y + Z\) is false and \(X < Z\): merge \(X\) and \(Y\), otherwise \(Y\) and \(Z\)

\[
\begin{array}{c|c|c|c}
\text{runLen} & 120 & 80 & 45 \\
\end{array}
\]
Our work: Testcase and worst-case analysis

Wrote program that generates testcase

- Exploits breaking the invariant by generating too many “short” runs
- Triggers error: insufficient size for runLen to store run lengths

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 40
    at java.util.TimSort.pushRun(TimSort.java:386)
    at java.util.TimSort.sort(TimSort.java:213)
    at java.util.Arrays.sort:Array.sort:659)
    at TestTimSort.main(TestTimSort.java:18)
```
Our work: Testcase and worst-case analysis

Wrote program that generates testcase

- Exploits breaking the invariant by generating too many “short” runs
- Triggers error: insufficient size for `runLen` to store run lengths

<table>
<thead>
<tr>
<th>Language</th>
<th>Smallest array that triggers error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android</td>
<td>65.536 ((2^{16}))</td>
</tr>
<tr>
<td>Java</td>
<td>67.108.864 ((2^{26}))</td>
</tr>
<tr>
<td>Python</td>
<td>562.949.953.421.312 ((2^{49}))</td>
</tr>
</tbody>
</table>

Most powerful supercomputer (Tianhe-2) has \(\approx 2^{50}\) bytes of mem.

Provided worst-case analysis of broken version

- Shows the actual minimally required `runLen.length`
Our work: Fixed algorithm

Fixed the algorithm

- Check that last 4 runs satisfy invariant
- Executed existing benchmarks (result: same performance) and unit tests (all passed)

```java
/** ... 
* merges adjacent runs until the stack invariants are reestablished:
* 1. runLen[i - 3] > runLen[i - 2] + runLen[i - 1]
* 2. runLen[i - 2] > runLen[i - 1]
*/
private void mergeCollapse() {
    while (stackSize > 1) {
        int n = stackSize - 2;
        if ( (n >= 1 && runLen[n-1] <= runLen[n] + runLen[n+1]) ||
            (n >= 2 && runLen[n-2] <= runLen[n-1] + runLen[n]) ) {
            if (runLen[n - 1] < runLen[n + 1])
                n--;
            } else if (runLen[n] > runLen[n + 1]) {
                break; // Invariant is established
            }
            mergeAt(n);
        }
    }
```
Analyzing “Real” Software

“because truly understanding it essentially requires doing a formal correctness proof, it’s difficult to maintain”

“Yet another large mass of difficult code can make for a real maintenance burden after I’m dead”

- Tim Peters on Timsort, python-dev mailing list, 2002

Implementation uses features for performance that complicate analysis: break statements, low-level bitwise ops., arithmetic overflows
Analyzing “Real” Software

“because truly understanding it essentially requires doing a formal correctness proof, it’s difficult to maintain”

“Yet another large mass of difficult code can make for a real maintenance burden after I’m dead”

- Tim Peters on Timsort, python-dev mailing list, 2002

Implementation uses features for performance that complicate analysis: break statements, low-level bitwise ops., arithmetic overflows

Mechanically proved fixed version with KeY

- Deductive Java theorem prover developed by TU Darmstadt (Reiner Haehnle), Karlsruhe, Chalmers
- termination, absence of the bug, and all other run-time exceptions
- this requires: formal specifications for all functions
Evaluation

Proof Stats - summary

<table>
<thead>
<tr>
<th></th>
<th># Rule Apps</th>
<th># Interactive</th>
<th>LoSpec</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>total</td>
<td>2,966,522</td>
<td>28,291 (0.95%)</td>
<td>526</td>
<td>536</td>
</tr>
<tr>
<td>w/o mergeLo</td>
<td>1,510,604</td>
<td>8,087 (0.54%)</td>
<td>460</td>
<td>442</td>
</tr>
</tbody>
</table>

Verification effort: ≈ 3 man-months
Evaluation

Proof Stats - summary

<table>
<thead>
<tr>
<th></th>
<th># Rule Apps</th>
<th># Interactive</th>
<th>LoSpec</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>total</td>
<td>2.966.522</td>
<td>28.291 (0.95%)</td>
<td>526</td>
<td>536</td>
</tr>
<tr>
<td>w/o mergeLo</td>
<td>1.510.604</td>
<td>8.087 (0.54%)</td>
<td>460</td>
<td>442</td>
</tr>
</tbody>
</table>

Verification effort: ≈ 3 man-months

Evaluation of the problem

- Bug unlikely to be encountered by accident
- Possible security hazard: bug may be exploitable in DoS attack
- Manual code reviews (Google) unable to expose bug
- Extensive testing unable to expose bug:
  - input size too large, structure too complex
- Widely used core libraries may contain subtle bugs for years
Java

- Submitted bug report to Java issue tracker
  https://bugs.openjdk.java.net/browse/JDK-8072909
Responses: developer communities

Java

- Submitted bug report to Java issue tracker
- Bug was previously found and "fixed" by increasing `runLen.length`

```java
int stackLen = (len < 120 ? 5 :
                len < 1542 ? 10 :
                len < 119151 ? 19 24
                                : 40);

runBase = new int[stackLen];
runLen = new int[stackLen];
```
Responses: developer communities

Java

- Submitted bug report to Java issue tracker
- Bug was previously found and “fixed” by increasing `runLen.length`
- Bug now fixed by further increasing `runLen.length` based on worst-case analysis

Discussion on OpenJDK mailing list

*Stack length increased previously by JDK-8011944 was insufficient for some cases. Please review and push* - Lev Priima, 11 Feb 2015

```java
int stackLen = (len < 120 ? 5 :
                   len < 1542 ? 10 :
                   len < 119151 ? 24 :
                                  40) 49);
runBase = new int[stackLen];
runLen = new int[stackLen];
```
Responses: developer communities - Java

Java

- Submitted bug report to Java issue tracker
- Bug was previously found and “fixed” by increasing `runLen.length`
- Bug now fixed by further increasing `runLen.length` based on worst-case analysis

Discussion on OpenJDK mailing list

Stack length increased previously by JDK-8011944 was insufficient for some cases. Please review and push
- Lev Priima, 11 Feb 2015

Hi Lev, The fix looks fine. Did you consider the improvements suggested in the paper to reestablish the invariant?
- Roger Riggs, Feb 11, 2015

Just briefly looked at it, w/o evaluating formal proof …
- Lev Priima, Feb 11, 2015
Java

- Submitted bug report to Java issue tracker
- Bug was previously found and “fixed” by increasing `runLen.length`
- Bug now fixed by further increasing `runLen.length` based on worst-case analysis
- Purported class invariant still broken
- Not amenable to mechanic verification
Responses: developer communities - Java

Java

- Submitted bug report to Java issue tracker
- Bug was previously found and “fixed” by increasing `runLen.length`
- Bug now fixed by further increasing `runLen.length` based on worst-case analysis
- Purported class invariant still broken
- Not amenable to mechanic verification
- **Early 2018: Error found in worst-case analysis!** “On the Worst-Case Complexity of TimSort” (ESA 2018) by N. Auger et al
Submitted bug report to Java issue tracker

Bug was previously found and “fixed” by increasing `runLen.length`

Bug now fixed by further increasing `runLen.length` based on worst-case analysis

Purported class invariant still broken

Not amenable to mechanic verification

Early 2018: Error found in worst-case analysis! “On the Worst-Case Complexity of TimSort” (ESA 2018) by N. Auger et al

July 2018: Java version fixed properly, by reestablishing invariant: https://bugs.openjdk.java.net/browse/JDK-8203864
Python

- Bug report filed by Tim Peters
- Bug fixed by checking last 4 runs (verified version)
Python
▶ Bug report filed by Tim Peters
▶ Bug fixed by checking last 4 runs (verified version)

Android
▶ Android v6: check last 4 runs
▶ Android v7 = OpenJDK
Responses: general public

- Popular articles (ERCIM, Bits & Chips)
- Blog post viewed 4499312 times
Responses: general public

- Popular articles (ERCIM, Bits & Chips)
- Blog post viewed 4499312 times

---

JetBrains IntelliJ IDEA Blog (March 10, 2015):
"The first of its kind, this result is both an important development for the Java community and a proof of concept for the feasibility of formal verification and automated theorem proving. Perhaps more importantly, the tool used to detect and identify this bug is completely open source and available to try yourself."

---

```plaintext
Some researchers found an error in the logic of merge_collapses, explained here, and with corrected code shown in section 3.2:

This affects all current versions of Python. However, I marked the priority 'low' because, as the article also notes, there's currently no machine in existence with enough memory to hold an array large enough for a contrived input to trigger an overflow of the pending-runs stack.

It should be fixed anyway, and their suggested fix looks good to me.
```

---

Pascal Hartig
@pjasny

The bug found in the Python and Java implementations of Timsort was also present in this Haskell version: github.com/limbg/haskell-timsort...

---

Pascal Hartig
@pjasny

I wonder if the invariant could have been encoded in the type system, or if LiquidHaskell, Iris, Agda impis would have exposed this.

---

Stijn de Gouw
Timsort
Eindhoven, Nov 24, 2018
Conclusion

- Even core Java libraries contain subtle bugs for years
- Master projects on verifying and improving Java libraries possible!
Useful links

<table>
<thead>
<tr>
<th>Blog post</th>
<th><a href="http://tinyurl.com/timsort-bug">http://tinyurl.com/timsort-bug</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Website with full paper, test programs and proofs</td>
<td><a href="http://www.envisage-project.eu/timsort-specification-and-verification">http://www.envisage-project.eu/timsort-specification-and-verification</a></td>
</tr>
<tr>
<td>KeY (Java theorem prover)</td>
<td><a href="http://www.key-project.org">http://www.key-project.org</a></td>
</tr>
<tr>
<td>Timsort description</td>
<td><a href="http://bugs.python.org/file4451/timsort.txt">http://bugs.python.org/file4451/timsort.txt</a></td>
</tr>
<tr>
<td>OpenJDK dev discussion</td>
<td><a href="http://mail.openjdk.java.net/pipermail/core-libs-dev/2015-February/thread.html#31405">http://mail.openjdk.java.net/pipermail/core-libs-dev/2015-February/thread.html#31405</a></td>
</tr>
</tbody>
</table>